.UBAfiuba
FACULTAD DE INGENIERÍA

Integración de Bases de Datos Relacionales y NoSQL

Fecha de entrega: Última semana de clases

Formato: Trabajo grupal

Curso : Merlino

Integrantes:

Padrón	**Apellido y nombre**
110845	Cardoso Landaburu Juan Gabriel
110598	Almanza Lucia
109473	Ariel Folgueira
100648	Daniela Pareja
101305	Ignacio Latorre
95971	Agustin Cavo

Objetivo del Proyecto:

El objetivo de este proyecto fue comprender las diferencias prácticas entre bases de datos relacionales y NoSQL mediante el desarrollo de una aplicación web que interactúa con ambos tipos de bases de datos. Se buscó implementar una interfaz intuitiva para realizar operaciones CRUD (crear, leer, actualizar y eliminar) en ambas bases de datos, explorando los desafíos y ventajas que presenta cada tecnología en un contexto práctico.

Introducción:

En la actualidad, el manejo de datos es una parte fundamental del desarrollo de aplicaciones. Las bases de datos relacionales, como MySQL o PostgreSQL, son conocidas por su estructura y fiabilidad, mientras que las bases de datos NoSQL, como Firebase o MongoDB, ofrecen flexibilidad y escalabilidad para datos no estructurados. Este proyecto combina ambas tecnologías en una solución integrada para gestionar datos de un sistema de cine, destacando sus fortalezas y limitaciones en un entorno real.

Elección de las Tecnologías

Frontend: React.js

Para el desarrollo del frontend de nuestra aplicación, decidimos utilizar React.js, una de las bibliotecas de JavaScript más populares y ampliamente adoptadas en la actualidad. React se distingue por su enfoque en la creación de interfaces de usuario dinámicas y altamente reactivas, lo que permite a los desarrolladores construir aplicaciones interactivas de una manera eficiente y mantenible.

Una de las características clave de React es su sistema de componentes. Este enfoque modular permite dividir la interfaz en pequeños bloques de código reutilizables, lo que facilita la organización del desarrollo y la mejora de la mantenibilidad a largo plazo. Además, React permite la actualización eficiente de la interfaz de usuario mediante su Virtual DOM, lo que optimiza el rendimiento de la aplicación al minimizar los cambios directos en el DOM real.

React también es compatible con una amplia gama de herramientas y bibliotecas, lo que facilita la integración con otros frameworks y tecnologías, como Redux para la gestión del estado o React Router para el manejo de rutas dentro de la aplicación. Esta flexibilidad y adaptabilidad hacen de React una excelente elección para el frontend de aplicaciones web modernas.

Backend: Node.js con Express

Para el backend, optamos por Node.js junto con Express, un framework minimalista y flexible para aplicaciones web. Node.js es una plataforma basada en JavaScript que permite ejecutar código fuera del navegador, lo que lo hace ideal para la creación de aplicaciones web escalables y rápidas. Utilizamos Node.js por su capacidad para manejar múltiples conexiones simultáneas de manera eficiente gracias a su modelo de E/S no bloqueante y su naturaleza orientada a eventos, lo cual lo convierte en una opción excelente para aplicaciones que requieren alta concurrencia.

Por otro lado, Express es un framework que simplifica la creación de APIs RESTful, proporcionando herramientas y convenciones que facilitan el desarrollo de rutas, controladores y middleware. Express es ampliamente utilizado debido a su simplicidad y flexibilidad, lo que permite una fácil integración con diferentes bases de datos y sistemas. Además, su gran comunidad de desarrolladores y la disponibilidad de numerosos paquetes

en npm lo convierten en una opción robusta para construir APIs rápidas y eficientes.

En conjunto, Node.js y Express forman una stack tecnológica ligera y de alto rendimiento, que es perfecta para aplicaciones en tiempo real o de alta carga, como en este caso, donde se requieren interacciones constantes con bases de datos y actualizaciones en tiempo real en el frontend. Esta combinación también permite trabajar en JavaScript tanto en el frontend como en el backend, lo que mejora la coherencia del desarrollo y facilita el mantenimiento del código.

Bases de Datos

Para las distintas bases de datos requeridas para el trabajo, decidimos utilizar las siguientes:

Relacional: MySQL

Para la gestión de datos estructurados y relaciones entre entidades en nuestra aplicación, optamos por MySQL, un sistema de gestión de bases de datos relacional ampliamente reconocido y utilizado en la industria del software. MySQL se caracteriza por su madurez, robustez y rendimiento, lo que lo convierte en una opción confiable para almacenar datos de forma organizada y eficiente. En nuestra aplicación, MySQL maneja operaciones complejas relacionadas con el sistema de gestión de entradas de cine, como la administración de películas, salas, horarios y disponibilidad de asientos.

MySQL utiliza el lenguaje de consulta estructurado SQL (Structured Query Language), lo que permite realizar consultas complejas y gestionar datos de manera eficiente a través de relaciones entre tablas. Las relaciones entre los diferentes elementos del sistema, como las películas, las salas y las entradas, se gestionan de manera natural en una base de datos relacional como MySQL. Esta estructura relacional facilita la integridad de los datos, así como la implementación de restricciones de clave primaria y relaciones entre tablas, garantizando que los datos estén bien organizados y sean consistentes.

NoSQL: Firebase Firestore

Para la gestión de datos no estructurados, como los comentarios y calificaciones de los usuarios, seleccionamos Firebase Firestore, una base de datos NoSQL basada en documentos proporcionada por Google Firebase. Firestore destaca por su facilidad de configuración y su capacidad para gestionar grandes cantidades de datos en tiempo real.

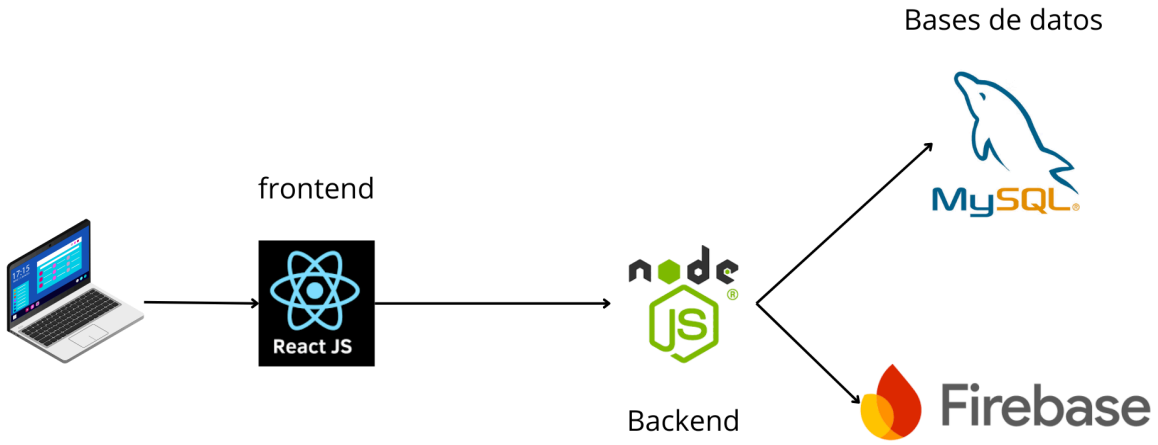
Esta característica es particularmente útil para aplicaciones que requieren actualizaciones dinámicas de datos sin necesidad de recargar la página, como los comentarios de los usuarios en nuestra plataforma.

FireStore permite almacenar y acceder a documentos JSON dentro de colecciones, lo que proporciona una estructura flexible que se adapta bien a datos que no siguen un esquema fijo. Esto hace que Firebase sea ideal para manejar información no estructurada como comentarios y calificaciones, donde cada documento puede tener diferentes campos y tipos de datos. Además, Firestore permite realizar consultas complejas sobre los documentos, aunque no necesariamente con la misma flexibilidad que un sistema relacional como MySQL.

Una de las principales ventajas de Firestore es su soporte en tiempo real, lo que permite a los usuarios ver los comentarios y calificaciones en tiempo real sin tener que recargar la página. Esta característica es crucial para aplicaciones interactivas como la nuestra, donde los usuarios deben poder ver las actualizaciones de los datos de manera instantánea.

Diagrama de Arquitectura

El sistema utiliza una arquitectura cliente-servidor en la que React.js actúa como frontend, y Node.js maneja las solicitudes hacia las bases de datos. A continuación, se presenta un diagrama de la arquitectura:



Configuración y Conexión a Bases de Datos

MySQL

Para conectar nuestra aplicación al sistema de base de datos relacional MySQL desde el backend, decidimos usar la librería mysql2, una opción robusta y eficiente para trabajar con bases de datos MySQL en aplicaciones Node.js. La configuración de la conexión se realizó utilizando las credenciales necesarias, como el host, usuario, contraseña y el nombre de la base de datos, que se almacenan de manera segura para evitar exponer datos sensibles en el código fuente.

1. Configuración de Conexión: Para establecer la conexión con la base de datos MySQL, se definieron los parámetros de configuración en un archivo de configuración del backend. Estos parámetros incluyen:
    - host: La dirección del servidor donde se encuentra la base de datos MySQL, que puede ser local o remota.
    - usuario: El nombre de usuario con permisos adecuados para acceder a la base de datos.
    - contraseña: La contraseña asociada al usuario para autenticar la conexión.
    - base de datos: El nombre de la base de datos a la que se desea acceder y realizar las operaciones.
2. Establecimiento de la Conexión: Utilizando la librería mysql2, establecemos una conexión de cliente que se configura para mantener la conexión abierta durante la duración de la aplicación. Esta conexión permite ejecutar consultas SQL de manera eficiente, como operaciones de inserción, actualización, eliminación y consulta sobre las tablas correspondientes a las películas, salas y entradas.
3. Manejo de Errores: En la configuración de la conexión, también se incluyó un manejo de errores básico que permite capturar y gestionar posibles fallos, como problemas con las credenciales de acceso o la falta de conexión al servidor de la base de datos. Esto asegura que la aplicación sea robusta y pueda reaccionar adecuadamente a errores de conexión.

Firebase Firestore

La conexión y configuración de Firebase Firestore en nuestra aplicación se realizó mediante la plataforma Firebase, un conjunto de herramientas que facilita la integración de servicios backend en aplicaciones web y móviles. Firebase Firestore es una base de datos

NoSQL que almacena los datos en documentos dentro de colecciones. La configuración para conectar nuestra aplicación al servicio Firestore se hizo de manera sencilla utilizando las credenciales generadas en la consola de Firebase.

1. Configuración Inicial: Para integrar Firebase, primero se generaron las credenciales de Firebase desde la consola de Firebase. Estas credenciales incluyen un archivo de configuración con claves específicas que permiten identificar nuestra aplicación y autenticar la conexión con Firestore de manera segura.
2. Conexión al Proyecto Firebase: Utilizamos la librería firebase/firestore proporcionada por Firebase. Esta librería es responsable de gestionar la conexión entre nuestra aplicación frontend y el servicio Firestore, permitiendo que las operaciones CRUD sobre los datos se realicen de manera eficiente. La conexión se establece en el frontend, lo que significa que las operaciones como la lectura y escritura de datos se hacen directamente desde la interfaz de usuario.
3. Operaciones CRUD en Firestore: Una vez configurada la conexión, utilizamos los métodos proporcionados por Firestore para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los documentos en la base de datos. Las colecciones de Firestore contienen documentos que son esencialmente objetos JSON, lo que permite manejar datos de manera flexible sin la necesidad de un esquema fijo. En nuestra aplicación, los comentarios de los usuarios y las calificaciones se almacenan en una colección específica, y cada nuevo comentario se guarda como un documento dentro de esta colección.
4. Actualizaciones en Tiempo Real: Firebase Firestore ofrece la ventaja de actualizaciones en tiempo real. Esto significa que cualquier cambio realizado en la base de datos (por ejemplo, agregar un nuevo comentario o modificar una calificación) se refleja de inmediato en la interfaz de usuario, sin necesidad de recargar la página. Para lograr esto, utilizamos los escuchadores en tiempo real proporcionados por Firestore, que permiten actualizar la vista cada vez que los datos cambian.
5. Manejo de Errores y Seguridad: Firebase también ofrece una capa de seguridad a través de las reglas de seguridad, que definen qué operaciones se pueden realizar en los datos, dependiendo de la autenticación del usuario. Aunque las reglas de seguridad predeterminadas son útiles durante el desarrollo, es importante personalizarlas según las necesidades específicas de la aplicación antes de la producción. Además, Firebase ofrece manejo de errores integrado que facilita la identificación y resolución de problemas de conexión o autorización.

Descripción de Funcionalidades CRUD

En nuestra aplicación, las funcionalidades CRUD (Crear, Leer, Actualizar, Eliminar) se implementaron para interactuar con ambas bases de datos, MySQL y Firebase Firestore, reflejando las diferencias clave entre un sistema relacional y una base de datos NoSQL.

Operaciones CRUD en MySQL

1. **Crear (Create):** Para insertar nuevos registros en la base de datos relacional, utilizamos comandos SQL `INSERT INTO`. En nuestro caso, esto permitió agregar entradas de películas al sistema, incluyendo información como el título de la película, la sala, la fecha, la hora y los asientos disponibles. Estas operaciones se ejecutan desde el backend a través de rutas API creadas con Express, que reciben los datos en formato JSON desde el frontend, los procesan y los almacenan en la base de datos.
2. **Leer (Read):** Las operaciones de lectura en MySQL se realizan con consultas SQL `SELECT`. Esto nos permitió recuperar datos específicos o listas completas de entradas, como todas las películas programadas en una sala. Las consultas incluyen filtros y condiciones según sea necesario (por ejemplo, buscar entradas por fecha o sala). Los resultados se envían al frontend en formato JSON para su visualización en la interfaz.
3. **Actualizar (Update):** Para modificar registros existentes, se utilizó la instrucción SQL `UPDATE`. Esto permitió ajustar datos como cambiar la hora de una función o actualizar el número de asientos disponibles. Las rutas API procesan las solicitudes desde el frontend, identificando el registro a actualizar mediante su ID único y aplicando los cambios enviados.
4. **Eliminar (Delete):** Para eliminar una entrada, usamos el comando SQL `DELETE`. Por ejemplo, si una función de película es cancelada, el registro correspondiente se elimina utilizando su ID único. Estas operaciones están protegidas para evitar eliminaciones accidentales o no autorizadas.

Diferencias Clave en MySQL:

- Estructura estrictamente definida mediante esquemas que especifican tipos de datos y relaciones entre tablas.

- Las operaciones requieren consultas SQL precisas, lo que facilita la integridad de los datos y las relaciones entre ellos.
- Ideal para gestionar datos estructurados y transacciones complejas, como las relacionadas con entradas y asientos.

Operaciones CRUD en Firebase Firestore

1. **Crear (Create):** En Firestore, los datos se almacenan como documentos JSON dentro de colecciones. Para agregar un nuevo comentario de usuario, utilizamos el método `addDoc` de la biblioteca de Firestore. Cada documento representa un comentario y contiene campos como el nombre de la película, el comentario del usuario y su calificación. No es necesario definir un esquema previamente, lo que permite mayor flexibilidad en los datos.
2. **Leer (Read):** La lectura de datos en Firestore se realiza utilizando el método `getDocs`, que permite recuperar todos los documentos de una colección o filtrarlos según criterios específicos. Esto se utilizó para mostrar en la interfaz una lista de comentarios asociados a cada película. Además, Firestore ofrece escuchadores en tiempo real (`onSnapshot`) que actualizan automáticamente los datos en el frontend si cambian en la base de datos.
3. **Actualizar (Update):** Para modificar documentos existentes, se utilizó el método `updateDoc`, que permite actualizar campos específicos de un documento. Por ejemplo, un usuario puede editar su comentario o cambiar la calificación asignada a una película. Los documentos en Firestore se identifican por un ID único generado automáticamente, que se utiliza para localizar el documento que se va a modificar.
4. **Eliminar (Delete):** La eliminación de documentos en Firestore se realiza mediante el método `deleteDoc`. Por ejemplo, si un usuario decide eliminar su comentario, el documento correspondiente se elimina de la colección `comentarios`. Esto elimina inmediatamente los datos y, gracias a la sincronización en tiempo real, la interfaz también refleja el cambio.

Diferencias Clave en Firestore:

- No utiliza esquemas fijos, lo que permite mayor flexibilidad en los datos almacenados.
- La estructura en forma de colecciones y documentos facilita el manejo de datos no estructurados, como los comentarios.

- Las actualizaciones en tiempo real mejoran la experiencia del usuario al reflejar cambios instantáneamente en la interfaz.
- Carece de soporte nativo para relaciones complejas, lo que lo hace menos adecuado para datos interrelacionados como los que se manejan en MySQL.

Comparación entre Bases de Datos Relacionales y NoSQL

Al trabajar con **MySQL** (relacional) y **Firebase Firestore** (NoSQL) en el desarrollo de la aplicación, identificamos varias ventajas y desventajas específicas de cada tipo de base de datos en función de sus características y su uso en este proyecto.

Ventajas de MySQL (Relacional)

- **Estructura Definida:** Su esquema bien definido asegura la consistencia de los datos, lo que es crucial para sistemas estructurados como la gestión de entradas y funciones de cine. Las relaciones entre tablas (por ejemplo, películas y asientos) permiten una organización clara y eficiente.
- **Consultas Complejas:** El lenguaje SQL facilita consultas complejas que involucran múltiples tablas, lo que resulta útil para análisis de datos o búsquedas específicas.
- **Transacciones:** MySQL soporta transacciones, lo que asegura la integridad de los datos en operaciones críticas, como la compra de boletos, evitando problemas como el "doble gasto".

Desventajas de MySQL:

- **Rigidez:** Requiere una estructura fija y un diseño detallado antes de la implementación, lo que puede limitar su adaptabilidad a cambios imprevistos.
- **Escalabilidad Horizontal Limitada:** Aunque MySQL puede escalar, hacerlo horizontalmente (agregar más servidores) es más complejo en comparación con sistemas NoSQL.

Ventajas de Firebase Firestore (NoSQL)

- **Flexibilidad de Datos:** No requiere un esquema rígido, lo que facilita el almacenamiento de datos diversos y no estructurados, como comentarios y calificaciones de usuarios. Los desarrolladores pueden añadir o modificar campos sin necesidad de reestructurar la base de datos.
- **Actualizaciones en Tiempo Real:** Firestore permite sincronización instantánea

entre el backend y el frontend, proporcionando una experiencia dinámica en la interfaz, especialmente útil para mostrar nuevos comentarios en tiempo real.

- **Escalabilidad:** Diseñado para manejar grandes volúmenes de datos y alto tráfico, Firestore es ideal para aplicaciones que crecen rápidamente o requieren disponibilidad global.

Desventajas de Firebase Firestore:

- **Falta de Relaciones Complejas:** No soporta relaciones nativas entre documentos, lo que dificulta gestionar datos interconectados como los necesarios en sistemas relacionales.
- **Costo:** A medida que la aplicación crece y se realizan más lecturas/escrituras, los costos de Firestore pueden aumentar significativamente en comparación con una base de datos relacional autogestionada.

Usos en Diferentes Contextos

1. **Cuando usar una Base de Datos Relacional (MySQL):**
    - **Sistemas Transaccionales:** Ideal para aplicaciones donde los datos deben ser consistentes y estructurados, como sistemas bancarios, gestión de inventarios o reservas.
    - **Análisis de Datos:** Adecuada para operaciones complejas, como generar reportes detallados o manejar grandes consultas con relaciones entre datos.
2. **Cuando usar una Base de Datos NoSQL (Firestore):**
    - **Datos Flexibles o No Estructurados:** Perfecta para aplicaciones con datos variables o no relacionados directamente, como comentarios de usuarios, logs o configuraciones personalizadas.
    - **Aplicaciones en Tiempo Real:** Fundamental para aplicaciones que requieren sincronización instantánea, como chats, tableros de anuncios o actualizaciones en vivo.

Ambos tipos de bases de datos tienen fortalezas y limitaciones que las hacen más adecuadas para diferentes escenarios. En el contexto de esta aplicación, MySQL proporcionó la solidez necesaria para gestionar datos estructurados y transacciones, mientras que Firestore ofrece flexibilidad y una experiencia de usuario fluida gracias a su capacidad de trabajar en tiempo real. Esta combinación permitió aprovechar lo mejor de ambos mundos, destacando la importancia de elegir la tecnología adecuada según los

requerimientos específicos del proyecto.

Dificultades

1. **Sincronización de datos:** Fue complicado asegurar que el backend comunicara correctamente las operaciones entre ambas bases de datos.
2. **Configuración de Firebase:** Inicialmente, las reglas de seguridad no permitían las operaciones CRUD desde el frontend.

Aprendizajes

- La elección de una base de datos depende del tipo de datos y del caso de uso.
- Aprender a poder hacer un frontend con buen diseño y que se pueda conectar con el backend
- La utilización de distintas herramientas para manejar sql, en este caso se usó phpAdmin.
- La sincronización entre frontend y backend, y lograr conectar las bases de datos para poder utilizarlas correctamente.

Conclusión

Este proyecto fue una valiosa oportunidad para explorar y comprender las diferencias prácticas entre bases de datos relacionales y NoSQL, así como los beneficios de integrarlas en un sistema que requiere gestionar distintos tipos de datos. Durante el desarrollo, experimentamos cómo las bases de datos relacionales como MySQL ofrecen una estructura rígida y robusta, ideal para operaciones que demandan consistencia y relaciones complejas, mientras que las bases de datos NoSQL como Firebase Firestore destacan en la flexibilidad y capacidad de manejar datos no estructurados en tiempo real.

La integración de ambas tecnologías demostró ser una solución efectiva para cubrir necesidades diversas dentro de la misma aplicación. Por ejemplo, MySQL fue crucial para garantizar la integridad de datos en la gestión de entradas y funciones de cine, mientras que Firestore proporcionó una experiencia de usuario dinámica y fluida al permitir la carga en tiempo real de comentarios y calificaciones. Esta combinación permitió crear un sistema versátil que no solo es funcional, sino también escalable y adaptable a futuras necesidades.

El desarrollo del proyecto presentó varios desafíos, como la configuración simultánea de

dos bases de datos con paradigmas diferentes, la implementación de operaciones CRUD que se ajustaran a sus respectivas características, y la sincronización eficiente entre el frontend, el backend y las bases de datos. Sin embargo, superar estas dificultades nos permitió consolidar nuestras habilidades en desarrollo full-stack, incluyendo la conexión de bases de datos, el diseño de arquitecturas de software híbridas y la optimización de la experiencia del usuario.

Además, este trabajo nos permitió reflexionar sobre la importancia de seleccionar la tecnología adecuada según los requisitos del proyecto. No existe una solución única que sea ideal para todos los casos; en su lugar, las fortalezas de cada tecnología deben aprovecharse en contextos específicos. Mientras que MySQL es indispensable para sistemas estructurados y críticos, Firestore demuestra su valor en aplicaciones flexibles y dinámicas. Esta perspectiva será clave para futuras decisiones de diseño en proyectos similares.